CSN08114

Scripting for Cybersecurity and Networks

Lecture 2: strings; lists; crypto

Today's Topics

- strings
- lists
- Encryption and decryption with the Caesar cipher

- Important concepts:
  - *Slicing*
  - *List comprehensions*

Python Object/Data Types

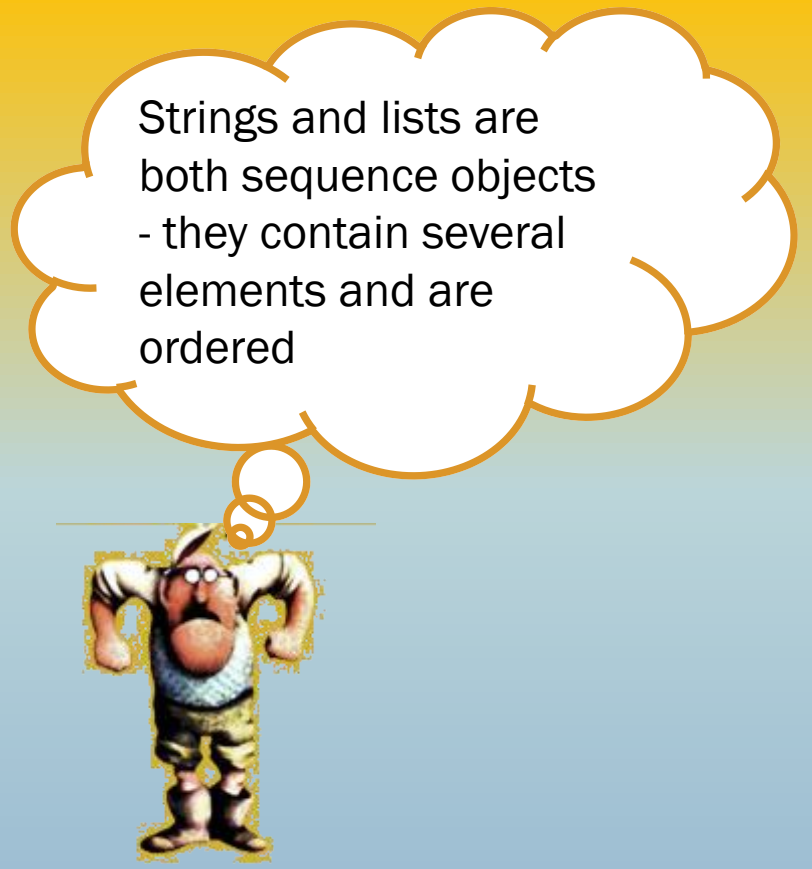
Everything in Python is an Object

Basic Data Types/Object Types:

- **Numbers:** `bool, int, float` ← Last Week
- **String:** `str`
- **Collection Objects:** ← This Week
  - ***List:*** `list`
  - ***Dictionary:*** `dict`
- **Tuple:** `tuple` ← Next Week

Go to www.menti.com

code **17 62 21**



Strings and lists

'CSN08x14'

[0, 'a', 'abc', [1,2,3]]

Strings

- a **String Object** is an ordered **Sequence** of characters in quotes.

```
str0 = ''                          # create empty string
str1 = 'parrot'                    # create string variable
str2 = "Don't Panic!! "            # can use " or '
str3 = """Don't Panic!!                  # multiline
and carry on"""
str4 = " Don't Panic!! \
and carry on"
```

- Can use " or ' and embed one within the other
- a single character is a string of length one (not a different data type)

Lists

- List is a *Sequence* of other objects
- Can contain different types of object – a *general* collection
- Ordered by position – same as strings – but elements not fixed size

Creation:

```
>>> list0 = []      #  empty list
>>> list1 = ['parrot', 4, -9.9, 'Knight']
```

List enclosed in [ ]

Difference between this and an C# or Java array?

More list Creation examples

```
>>> passwords = ['password', 'qwerty', 'default', [1234, 12345, 123456]]
```

List objects can be nested within another list

```
>>> matrix = [ [1, 2, 3],    # Offsets [0][0], [0][1], [0][2]
               [4, 5, 6],    # Offsets [1][0], [1][1], [1][2]
               [7, 8, 9]]    # Offsets [2][0], [2][1], [2][2]
```

Can span multiple lines – [ ] groups list together

```
>>> matrix
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Operators for sequence objects

+ *Concatenation operator*
joins strings/lists together to create a new object

```
>>> ['a',0]+['more','elements']
['a', 0, 'more', 'elements']
>>> 'dead '+'parrot'
'dead parrot'
```

***** *Repetition operator*

create new string or list by repeating another

```
>>> '2'*5
'22222'
>>> ['fun!']*3
['fun!', 'fun!', 'fun!']
```

---

in *Membership operators*

not in

```
>>> 'fun' in 'Python is fun'
True
>>> 'bored' not in ['Python','is','fun']
True
```

Slicing Operators: accessing the contents of strings and lists

Slicing operators return substrings from strings and elements from lists

`[offset]`

returns single character at the offset (counting from 0)
a **negative** offset is counted from the end of the string (counting from -1)

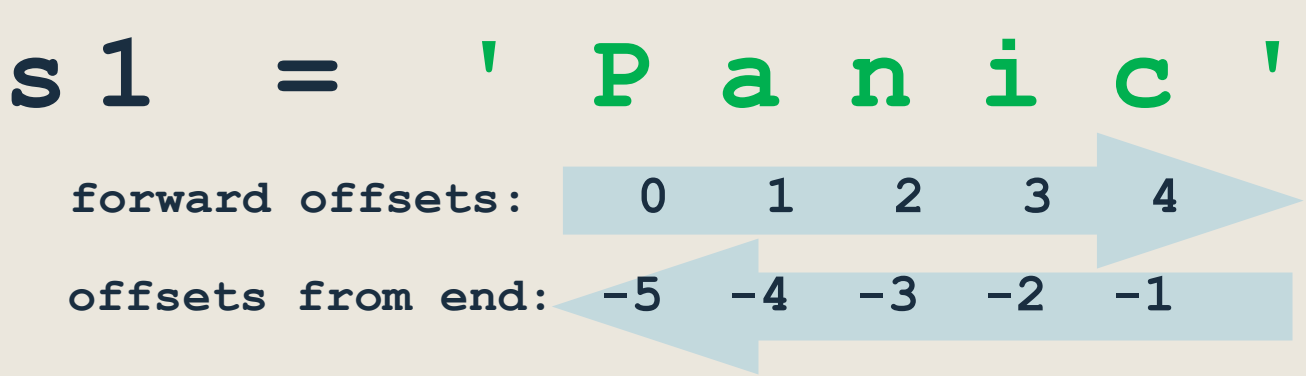
`[start:stop]`

returns **Slice** - substring of chars from start to stop
default start is 0
default end is last char in string
if start is greater or equal to stop the result is an empty string

`[start:stop:step]`

returns **Slice** - substring of chars from start to stop in steps of step
default step is 1. Negative step means go backwards

String Slicing examples



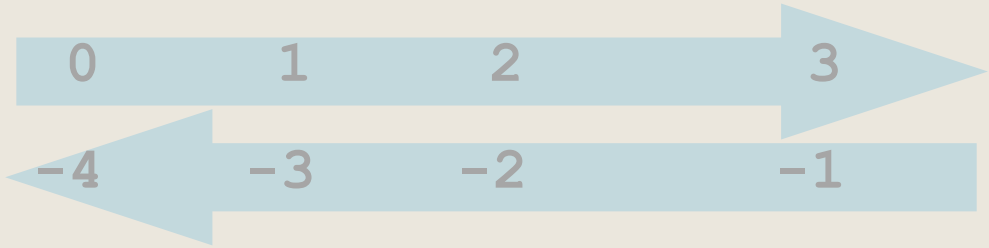
Used with – (minus sign), index and slice work backwards from end of string

```
>>> s1[0]       # Return char at offset 0
'P'
>>> s1[1:3]     # Slice offset 1 to (before) 3 (not incl)
'an'
>>> s1[1::2]    # Slice offset 1 to end in steps of 2
'ai'
>>> s1[-1]      # Slice last char
'c'
>>> s1[-3:]     # Slice last 3 chars
'nic'
```

List Slicing examples

```
list1 = ['parrot', 4, -9.9, 'Knight']
```

Offsets:




```
>>> list1[0]  # index by offset
'parrot'
>>> list1[-2:]    # slicing a list out of a list
[-9.9, 'Knight']
```

Slicing nested lists

```
>>> passwords = ['password', 'qwerty', 'default', [1234, 12345, 123456]]
>>> passwords[-1]
      [1234, 12345, 123456]
>>> passwords[-1][1:]
      [12345, 123456]

>>> matrix = [ [1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> matrix[1][2]
      6
>>> matrix[2]
      [7, 8, 9]
```

Mutable vs immutable objects

Lists are mutable objects – they CAN be changed

```
>>> list1 = ['parrot', 4, -9.9, 'Knight']
>>> list1[0] = 'squirrel'
>>> list1
['squirrel', 4, -9.9, 'Knight']
```

Replaces part of list -
Does NOT return new list object!

Strings are immutable objects - they cannot be modified

```
>>> str1 = 'abcde'
>>> str1[2]='Z'
Traceback (most recent call last):
  File "<pyshell#48>", line 1, in <module>
    str1[2]='Z'
TypeError: 'str' object does not support item assignment
```

Useful functions and methods

Useful functions for strings and lists

- Length: **`len()`**

```
>>> s = 'programming is fun'
>>> len(s)
18
```

```
>>> passwords = ['password', 'qwerty', 'default', [1234, 12345, 123456]]
>>> len(passwords)
4
```

- Object type: **`type()`**

```
>>> type(s)
<class 'str'>
>>> type(passwords)
<class 'list'>
```

- Check for a specific type: **`isinstance()`**

```
>>> isinstance(s,list)
False
>>> isinstance(s,str)
True
```

Object methods for strings and lists

Object Methods are attributes of an object

Methods are always appended to the object **with a dot**

Methods and functions perform a similar purpose but are applied differently

`.index()` and `.count()` work for both strings and lists

All other methods are specific and work only either for strings or for lists

`.count()`

counts the number of matching items

```
>>> 'parrot'.count('x')
0
>>> 'parrot'.count('r')
2
>>> list2
['Python', 'fun', 'is']
>>> list2.count('is')
1
```

`.index()`

Returns the position of the first matching item

```
>>> list2.index('is')
2
>>> 'parrot'.index('r')
2
>>> 'parrot'.index('x')
Traceback (most recent call last):
  File "<pyshell#100>", line 1, in <module>
    'parrot'.index('x')
ValueError: substring not found
```

String object methods

There are many methods that work only for strings

Examples:

- Case conversion: **`.upper()` `.lower()` `.capitalize()` `.title()`**
- Case testing: **`.isupper()` `.islower()`**
- Test for letter or number: **`.isalpha()` `.isdigit()`**

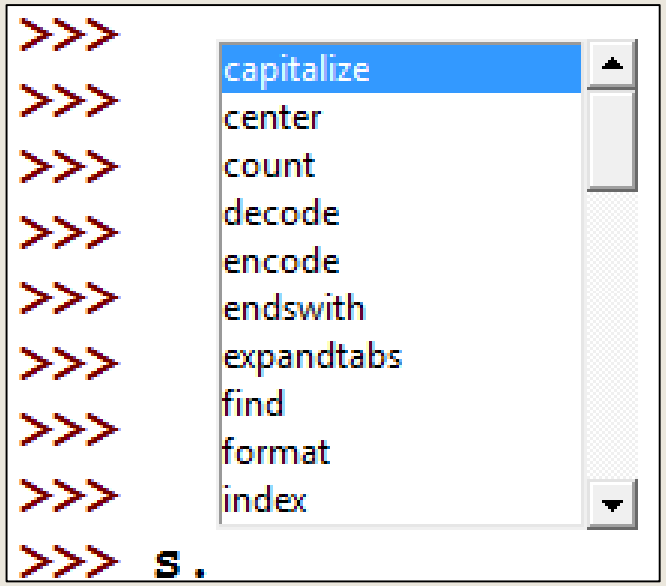
- **`.find()`** is similar to **`index()`** but returns -1 when not found

```
>>> 'parrot'.find('p')
0
>>> 'parrot'.find('P')
-1
```

```
>>> 'parrot'.upper()
'PARROT'
>>> 'parrot'.upper().isupper()
True
>>> 'parrot 123'[-3:].isdigit()
True
```

more String object methods

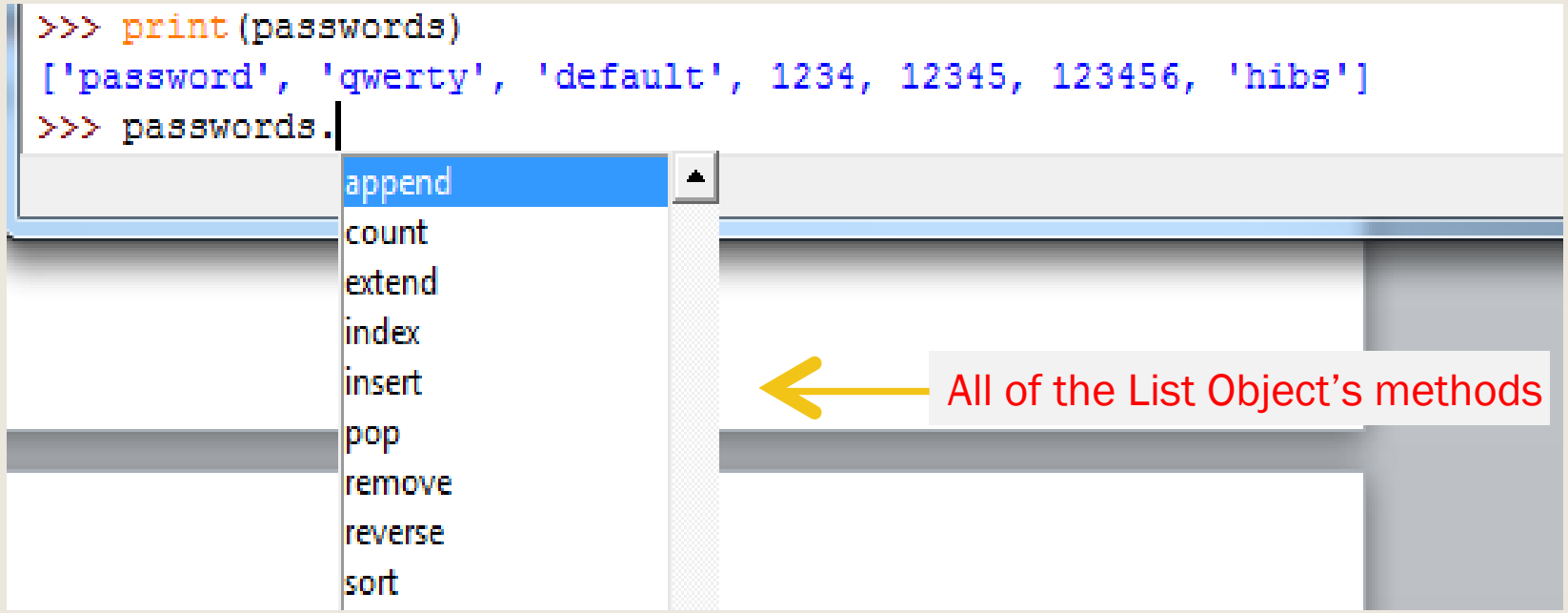
- To check which methods are available for any object, type
  - the name of the
  - **`Object.`**
  - Followed by a dot
  - Then wait or type
  - **<CTRL+SPACE>**



- Use up and down arrow keys
- to select a method

List Specific Methods

- Methods of the list object can change the list unlike the common sequence operators – mutable sequence
- List Methods from IDLE: Listname.<CTRL+SPACE>



What might append and insert do?

Methods to add to a list: append, insert, extend

```
>>> L = [1, 5, 2, 'parrot']
>>> L.append('hack')    # add object to list end - single object only
>>> L
 [1, 5, 2, 'parrot', 'hack']

>>> L.extend(['ing','is','bad'])    # add multiple objects to end
>>> L
 [1, 5, 2, 'parrot', 'hack', 'ing', 'is', 'bad']

>>> L.insert(1, 'knight')     # insert at given offset
>>> L
 [1, 'knight', 5, 2, 'parrot', 'hack', 'ing', 'is', 'bad']
```

Removing from a List Object: remove, pop, del

```
>>> l = [1, 2, 5, 'parrot', 'hack']
```

Remove generates error if value not found

```
>>> l.remove('hack')    # remove by value (removes first occurrence only)
>>> l
 [1, 2, 5, 'parrot']

>>> l.pop(1)  # remove by index and return removed value
 2
>>> l
 [1, 5, 'parrot']
```

The command **del** can be used to delete any object, and to delete elements from a list

```
>>> s1='abcd'
>>> list3=[2,4,5,6]
>>> del s1
>>> del list3[2]
>>> list3
[2, 4, 6]
>>> del list3
>>> list3
Traceback (most recent call last):
  File "<pyshell#121>", line 1, in <module>
    list3
NameError: name 'list3' is not defined
```

Sorting lists

Only works if the elements of the list can be compared using "<"

- **`sorted()`** function

```
>>> sorted(['python','is','fun'])
['fun', 'is', 'python']
```

```
>>> sorted('parrot',reverse=True)
['t', 'r', 'r', 'p', 'o', 'a']
```

Functions return a new object. The original list is not changed

- **`list.sort()`** method

```
>>> list2
['Python', 'is', 'fun']
>>> list2.sort()
>>> list2
['Python', 'fun', 'is']
```

The sort() method changes the list in place.
Doesn't work for strings - they are immutable

Python List Copying

- [] Operator can be used to copy lists

```
>>> list1 = [1, 2, 5, 'parrot']
>>> list2 = list1[:]
>>> list1[1] = 4
>>> print (list1, list2)
    [1, 4, 5, 'parrot'] [1, 2, 5, 'parrot']

>>> list3 = list1
>>> list1[1] = 3
>>> print (list1, list3)
    [1, 3, 5, 'parrot'] [1, 3, 5, 'parrot']
```

Using [:], we create a copy of the list
(2 variables pointing at different lists)

But without using [] we get
2 variables pointing at same stored list
Here, list3 and list1 will ALWAYS remain identical

Iterating (looping) over strings and lists
Incl list comprehension

Loops on strings

A string is a sequence that can be iterated over!!

```
>>> for ch in 'xyz':
        print(ch)


x
y
z
```

ch is assigned each of the characters of the string 'xyz', one at a time, as we iterate through the string

```
>>> string='abc'
>>> for ch in string:
        print(f'letter {string.index(ch)}: {ch}')


letter 0: a
letter 1: b
letter 2: c
```

List iteration: looping through a list

- Lists are iterables, so we can loop through a list

```
for item in list:
    # do something with item
```

```
>>> squares = [1, 4, 9, 16]
>>> sum=0
>>> for num in squares:
        sum+=num
        print(f'+ {num} = {sum}')


+ 1 = 1
+ 4 = 5
+ 9 = 14
+ 16 = 30
```

num – a variable which is automatically assigned to each item in list we are looping over

Block of code using each num value

List Comprehension

- Shorthand for list iteration
- A **List Comprehension** creates new list by performing an expression on items in an existing list

Syntax:

```
list = [<do something with> item for item in list]
```

- List comprehensions are much more efficient than an equivalent list iteration
- Can express complex loops, with conditions, as a list comprehension
- See e.g. http://treyhunner.com/2015/12/python-list-comprehensions-now-in-color/ for explanation and animated examples.

List Comprehension - example 1

```
>>> nums = [1,2,3,4]
>>> squares = [ n*n for n in nums]
>>> squares
[1, 4, 9, 16]
```

n – variable which is automatically assigned to each item in List we are looping over

Expression using each num value

List Comprehension - examples 2

```
>>> matrix = [ [1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]]


>>> last = [ x[-1] for x in matrix]
>>> last
  [3, 6, 9]
>>> flat = [n for row in matrix for n in row]
>>> flat
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

range()

- Range() is a BIF that generates numeric sequences
- Outputs list of integers (integer range), based on start, stop, and step arguments

seq_list = range([start,] stop[, step])

```
>>> range(5)    # stop at index 5, start defaults to 0
[0, 1, 2, 3, 4]

>>> range(5, 10)     # start at 5, stop at 10
[5, 6, 7, 8, 9]

>>> range(0, 10, 3) # increment of 3
[0, 3, 6, 9]
```

range() examples in for loops

What are the results of the following?

```
>>> for n in range(10,13):
        print(n)
```

```
>>> for n in range(10,20,5):
        print(n)
```

Start Stop Step (optional)

List Comprehension - example 3

- List comprehensions can contain if statements
- Can create list on the fly with range() function

```
>>> print(", ".join(["ha" if i else "Ha" for i in range(3)]) + "!")

Ha, ha, ha!
```

(from: https://stackoverflow.com/questions/4260280/if-else-in-pythons-list-comprehension)

Conversion and formatting

Converting strings to lists,
flattening lists,
formatting strings and numbers (e.g. for printing)

str()

- `str()` converts an object to a string
- Usually used for numbers
  - *Necessary because Python uses strict typing and does not convert implicitly*

```
>>> s = 'combos for 8 char password: ' + 95**8
Traceback (most recent call last):
  File "<pyshell#71>", line 1, in <module>
    s = 'combos for 8 char password: ' + 95**8
TypeError: must be str, not int
```

```
>>> s = 'combos for 8 char password: ' + str(95**8)
>>> s
'combos for 8 char password: 6634204312890625'
>>> 95**8
```

join() and split()

- **Convert a List to a String**

```
>>> '<sep char>'.join(<list>)
```

- **Convert String into a List**

```
>>> string.split('<separation char>')
```

You can also use **list(<my_string>)** to convert a string into a list of single letters

```
>>> list('abc')
['a', 'b', 'c']
```

- **Examples:**

```
>>> list1 = ['aaa', 'bbb', 'c', 'dd']
>>> s = ','.join(list1); s
'aaa,bbb,c,dd'
```

join() often used to Convert to string for printing

```
>>> newList = s.split(','); print (newList)
['aaa', 'bbb', 'c', 'dd']
```

Formatting numbers for output

- To give numbers a specific format when printing, we can use format placeholders. Can be used with .format, f-strings etc.
- General syntax: [flags][width][.precision]type
- See following slides for context

Example: z=123.456789	
Format placeholder	result
6.2f	'123.46'
6.0f	' 123'
+6.0f	' +123'
6.1e	'1.2e+02'
6.4e	'1.2346e+02'

Example: y=1234	
Format placeholder	result
2d	'1234'
8d	' 1234'
x	'4d2'
#0x	'0x4d2'
b	'10011010010'

Types
d - decimal
f - float
e - exponential
(scientific notation)
x - hex
b - binary

Formatting strings for output (printing)

2 recommended methods (+2 clumsy/legacy)

Can be used in many places but most often found in print statements

- **Literal string interpolation ('f'-strings)**
  streamlined, simplified string formatting (new for Python 3.6)
- **.format()** method
  very powerful and flexible but more cumbersome

- **Concatenation**
  easy but a bit clumsy
- **"%" method**
  a legacy method; should not be used in new code.

Literal string interpolation ("f-strings")

- Introduced with Python 3.6
- Aim to provide a method for formatting strings that is simpler and more streamlined than .format() but solves the problems of the legacy % method. Seems faster than .format().
- See https://www.python.org/dev/peps/pep-0498/

```
>>> lang='Python'
>>> many=1
>>> print(f'{lang} has {many} way to do things')
Python has 1 way to do things
```

```
>>> value=80
>>> f'The value is {value}.'
'The value is 80.'
>>> value = [90.65]
>>> f'The value is {value}.'
'The value is [90.65].'
>>> value = [90, 'abc']
>>> f'The value is {value}.'
"The value is [90, 'abc']."
```

.format() method

- Introduced with Python 2.6
- Very powerful and flexible - takes a bit of practice!
- Can be used directly with dictionaries (→ later)

```
>>> 'Capital of {}: {}'.format('Canada', 'Ottawa')
'Capital of Canada: Ottawa'
>>> '{1} is the capital of {0}'.format('Canada', 'Ottawa')
'Ottawa is the capital of Canada'
>>> '{0:d} decimal is {0:X} Hex and {0:b} binary'.format(100)
'100 decimal is 64 Hex and 1100100 binary'
```

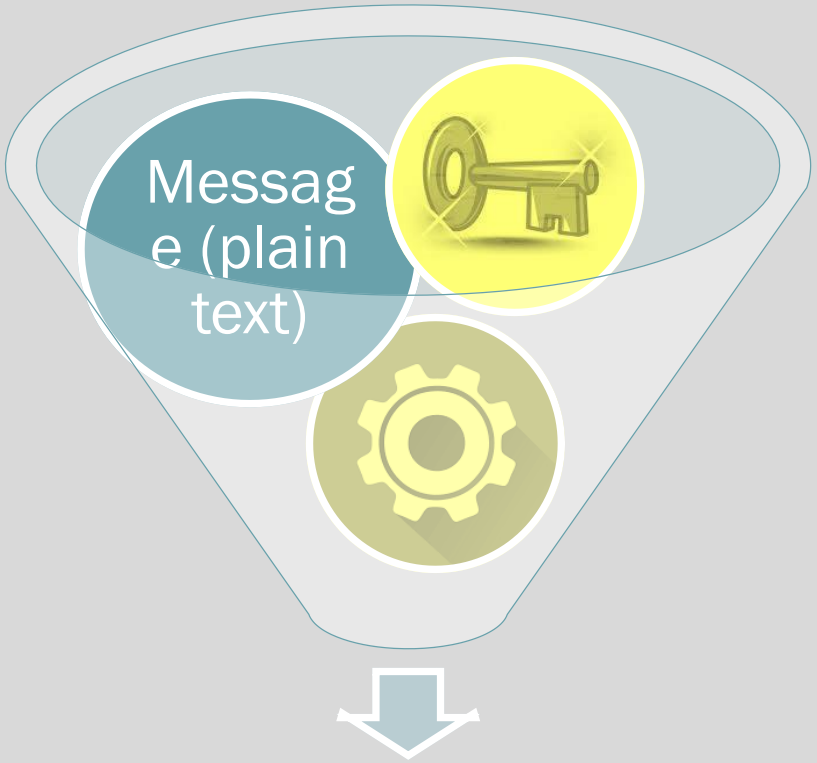
Here 0 and 1 refer to the zeroth and first variable given in the format() method

See http://www.python-course.eu/python3_formatted_output.php

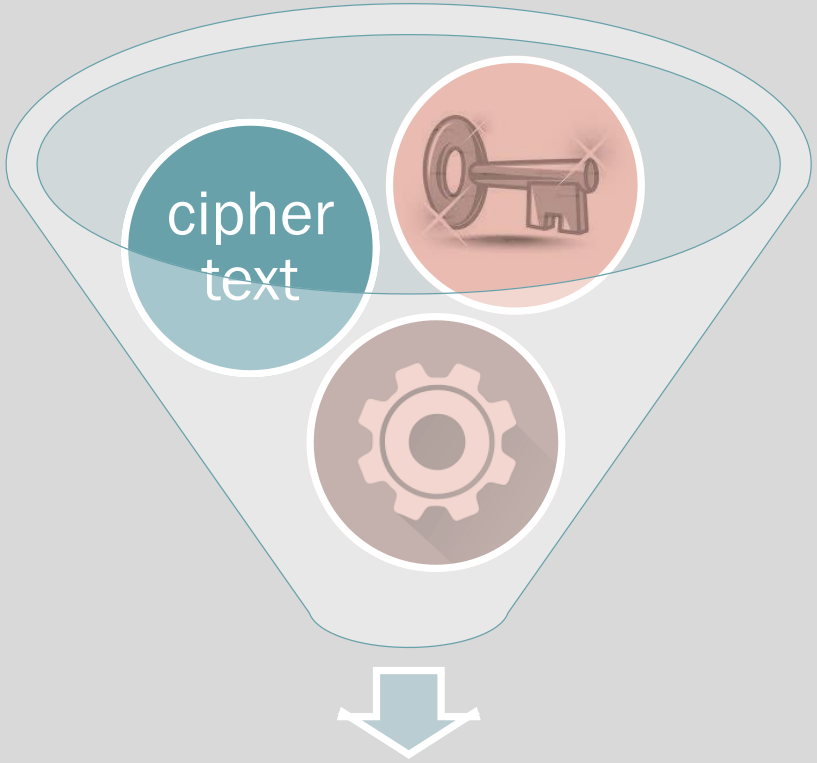
Cybersecurity: Cryptography

What is Cryptography?

Encryption and Decryption



ciphertext



Message (plaintext)

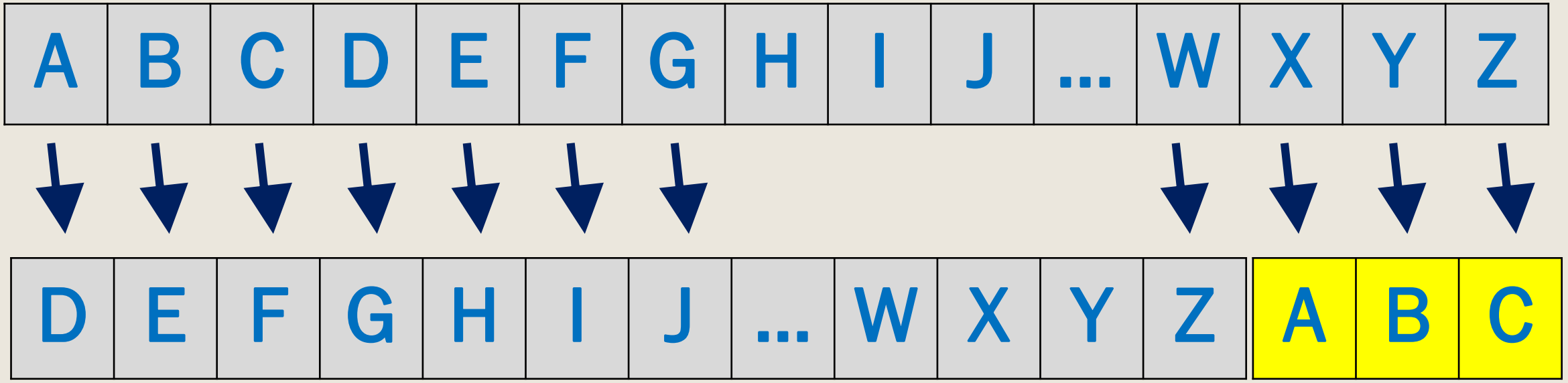
Caesar cipher

- One of the earliest documented encryption methods (Julius Caesar 100-44BC)
- A shift cipher: Shift each letter x characters to the right
- Simple algorithm – but initially very effective
- Broken 9th century through frequency analysis (Al-Kindi)

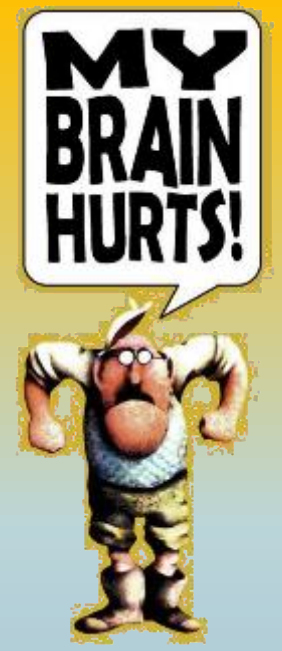
- https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/caesar-cipher

Caesar Cipher with Shift 3

- Shift each letter "3 to the right" in the alphabet



- "wrap round" back to the beginning of the alphabet
- To decipher – shift 3 **to the left** and wrap round



Go to www.menti.com and use code 17 62 21

Practical Lab 02

Coding the Caesar cipher

Coding the Caesar cipher

- **caesar_start.txt** gives you a template – this should run but not do much

```
====== RESTART: C:\Users\Petra\Dropbox\CSN08114 Python\caesar_start.py =
[*] ENCRYPTING - key: 2; plaintext: Hello Suzanne
[*] ciphertext: ello uzanne
[*] DECRYPTING - key: 2; ciphertext: IQQfOQtpKpIGXGtaQPG
[*] plaintext:
```

- What would we need to know to complete the module?

Coding the Caesar cipher

What do we need to know to complete the module?

3 distinct aspects, tackle one by one:

- How to encrypt?
- How to decrypt?
- How to crack?

Coding the Caesar cipher: front matter

1. Start by listing the characters we want to be able to encrypt
   - *E.g. the alphabet, capital letters A-Z, …*
2. To "wrap round", need to know how many characters there are

```
# Script:  caesar.py
# Desc:    encrypt and decrypt text with a Caesar cipher
#          using defined character set with index
# Author:  Petra Leimich
# Created: 23/9/17
# note that you should add a module doc string!

①  charset="ABCDEFGHIJKLMNOPQRSTUVWXYZ" # characters to be encrypted
②  numchars=len(charset) # number of characters, for wrapping round
```

Coding the Caesar cipher: Encrypt

1. Convert to upper case: use the text.upper() method
2. New character: look up in charset, wrap around if necessary

```
def caesar_encrypt(plaintext,key):
    """put an appropriate function doc string here"""
    print (f'[*] ENCRYPTING - key: {key}; plaintext: {plaintext}')

① # plaintext=     # convert plaintext to upper case
    ciphertext=''    # initialise ciphertext as empty string

    for ch in plaintext:
        if ch in charset:
②           new='' # replace this with your code, may use extra lines
        else:
            new=ch # do nothing with characters not in charset
        ciphertext=ciphertext+new
    print (f'[*] ciphertext: {ciphertext}')
    return ciphertext # returns ciphertext so it can be reused
```

Coding the Caesar cipher: Encrypt

①
```
charset="ABCDEFGHIJKLMNOPQRSTUVWXYZ" # characters to be shifted
numchars=len(charset) # number of characters, for wrapping round
```

Look up the position of ch in the charset

Using .find() method

```
>>> charset.find('A')
0
>>> charset.find('M')
12
```

```
>>> pos=charset.find('M')
>>> ch='M'
>>> pos=charset.find(ch)
>>> pos
12
```

Coding the Caesar cipher: Encrypt

```
charset="ABCDEFGHIJKLMNOPQRSTUVWXYZ" # characters to be shifted
numchars=len(charset) # number of characters, for wrapping round
```

② Shift position by the key and

③ look up character at that position

```
>>> pos
12
>>> key=2
>>> pos=pos+key
>>> pos
14
```

```
>>> charset[pos]
'O'
```

Coding the Caesar cipher: Encrypt

```
charset="ABCDEFGHIJKLMNOPQRSTUVWXYZ" # characters to be shifted
numchars=len(charset) # number of characters, for wrapping round
```

What if the shift takes us past the end of the charset?

```
>>> ch='Z'
>>> charset.find(ch)+key
27
>>> charset[charset.find(ch)+key]

Traceback (most recent call last):
  File "<pyshell#93>", line 1, in <module>
    charset[charset.find(ch)+key]
IndexError: string index out of range
```

Coding the Caesar cipher: Encrypt

```
charset="ABCDEFGHIJKLMNOPQRSTUVWXYZ" # characters to be shifted
numchars=len(charset) # number of characters, for wrapping round
```

④

Wrapping around.
Here 26 should become 0, 27 → 1, 28 → 2, etc

Try: pos=pos-26

we could subtract, but only

if pos >= len(charset)

```
>>> 26 - numchars
0
>>> 27 - numchars
1
>>> 4 - numchars
-22
>>> 56 - numchars
30
```

Coding the Caesar cipher: Encrypt

```
charset="ABCDEFGHIJKLMNOPQRSTUVWXYZ" # characters to be shifted
numchars=len(charset) # number of characters, for wrapping round
```

④

Wrapping around. Here 26 should become 0, 27 → 1, 28 → 2, etc

Better solution:

use modulo % operator!

pos = pos % numchars

```
>>> 26 % numchars
0
>>> 27 % numchars
1
>>> 4 % numchars
4
>>> 56 % numchars
4
```

Coding the Caesar cipher: Decrypt

```
charset="ABCDEFGHIJKLMNOPQRSTUVWXYZ" # characters to be shifted
numchars=len(charset) # number of characters, for wrapping round
```

② Subtract to shift position to the left

```
>>> pos = 14
>>> pos = pos-key
>>> pos
12
```

Everything else remains same, except input is ciphertext and output is plaintext!

How to crack?

What if we have ciphertext and don't know the key???

Brute force cracking:

- Decrypt with every possible key in turn
- Human can then inspect results and pick out the right one

```
>>> caesar_crack('IQQfOQtpKpIGXGtaQPG')
[*] CRACKING: IQQfOQtpKpIGXGtaQPG
Key:  0:  Result:  IQQFOQTPKPIGXGTAQPG
Key:  1:  Result:  HPPENPSOJOHFWFSZPOF
Key:  2:  Result:  GOODMORNINGEVERYONE
Key:  3:  Result:  FNNCLNQMHMFDUDQXNMD
Key:  4:  Result:  EMMBKMPLGLECTCPWMLC
Key:  5:  Result:  DLLAJLOKFKDBSBOVLKB
Key:  6:  Result:  CKKZIKNJEJCARANUKJA
Key:  7:  Result:  BJJYHJMIDIBZQZMTJIZ
```

How to crack?

Brute force cracking: Decrypt with every possible key in turn

```
def caesar_crack(ciphertext):
    """put an appropriate function doc string here"""
    # how could you brute force crack a caesar cipher?
    # your code here
```

- loop that iterates through range(numchars)
- Something like:

crack = caesar_decrypt(ciphertext,x)

print crack

To be able to use the result of caesar_decrypt(), the function needs to have a return statement:

```
return plaintext # returns plaintext so it can be reused
```